DAWA

Home About Content Others

Arquitectura de Aplicaciones WEB

VICTOR CHALEN

Definición

La arquitectura de aplicaciones web se refiere al diseño estructural y organizacional de los componentes que forman una aplicación web. Este diseño determina cómo se interconectan y comunican los distintos elementos de la aplicación para lograr que funcione de manera óptima, cumpla con los requerimientos funcionales y no funcionales, y se mantenga escalable y segura.

Arquitectura de Aplicaciones Web

Conceptos Clave

- Cliente/Servidor: El modelo fundamental de una aplicación web donde el cliente (navegador o aplicación en un dispositivo) solicita información y el servidor la procesa y responde.
- Capas: Las aplicaciones web suelen estar divididas en varias capas lógicas (presentación, lógica de negocio y datos) para separar responsabilidades y facilitar el mantenimiento.
- Escalabilidad: La capacidad de una aplicación para manejar un incremento en la carga de trabajo sin comprometer el rendimiento.
- Disponibilidad: Asegura que la aplicación esté accesible para los usuarios la mayor parte del tiempo.
- Seguridad: Implementa medidas para proteger la aplicación y los datos de los usuarios.

Componentes básicos de una aplicación web

Frontend (Cliente):

- Es la interfaz de usuario, el componente que interactúa directamente con el usuario. Se desarrolla principalmente con tecnologías como HTML, CSS y JavaScript. El frontend puede ser un simple sitio web estático o una aplicación compleja de una sola página (SPA, por sus siglas en inglés).

Backend (Servidor):

- El backend se encarga de la lógica de negocio de la aplicación. Es responsable de procesar las solicitudes del cliente, ejecutar operaciones, y devolver las respuestas. Desarrollado en varios lenguajes y frameworks como Node.js, Python (Django, Flask), Java (Spring), entre otros.

Base de datos:

- Donde se almacena la información persistente de la aplicación, como datos de usuario y contenido. Existen bases de datos relacionales (SQL) como PostgreSQL o MySQL y no relacionales (NoSQL) como MongoDB.

Tipos de Arquitectura

1. Arquitectura Monolítica

En una arquitectura monolítica, todos los componentes de la aplicación están integrados en una única unidad o servidor. El frontend, backend y base de datos funcionan juntos como un solo sistema.

Ventajas:

- Simplicidad en el desarrollo y despliegue.
- Menor complejidad en la comunicación interna.
- Buena elección para aplicaciones pequeñas o con requerimientos sencillos.

Desventajas:

- Dificultad para escalar de manera independiente.
- Riesgo de que un fallo afecte toda la aplicación.
- Actualizaciones y despliegues más lentos.

Tipos de Arquitectura

2. Arquitectura de Microservicios

En una arquitectura de microservicios, la aplicación se divide en múltiples servicios pequeños y autónomos, cada uno responsable de una funcionalidad específica. Estos servicios se comunican a través de APIs.

Ventajas:

- Escalabilidad independiente de cada servicio.
- Mayor resiliencia, ya que los fallos en un servicio no afectan toda la aplicación.
- Flexibilidad en la elección de tecnologías para cada microservicio.

Desventajas:

- Mayor complejidad en la gestión y comunicación entre servicios.
- Requiere mayor inversión en infraestructura y monitoreo.
- Más difícil de implementar en aplicaciones con equipos pequeños o sin experiencia en microservicios.

DAWA

Home About Content Others

Patrones de Arquitectura Web

Definición

Los patrones de arquitectura web son esquemas o estructuras de diseño que ayudan a organizar el código de una aplicación de manera eficiente, haciendo que sea más fácil de mantener, escalar y probar. Cada uno de estos patrones se utiliza para manejar la interacción entre las diferentes capas y componentes de una aplicación, asegurando una separación clara de responsabilidades.

Patrones de Arquitectura Web

1.Modelo Vista Controlador (MVC)

Es uno de los patrones más utilizados en aplicaciones web. Se divide en tres componentes:
Modelo: Representa la lógica de negocio y el acceso a datos.
Vista: Es la interfaz de usuario; muestra los datos del modelo y envía los comandos del usuario al controlador.
Controlador: Actúa como intermediario, procesando las entradas del usuario, actualizando el modelo, y seleccionando la vista adecuada para presentar la información.
Ventajas:

- Separación de responsabilidades, facilitando el mantenimiento.
- Permite trabajar en diferentes capas sin interferir con las otras.

Desventajas:

- Puede añadir complejidad en proyectos pequeños.
- La dependencia del controlador puede ser un cuello de botella en aplicaciones grandes.

2.Modelo Vista VistaModelo (MVVM)

MVVM es una evolución de MVC que introduce el "ViewModel" para gestionar la lógica de presentación y la comunicación entre el modelo y la vista.
Modelo: Gestiona los datos y la lógica de negocio.
Vista: Representa la interfaz de usuario.
VistaModelo: Es una representación de los datos del modelo que la vista puede utilizar directamente; facilita el enlace de datos o data binding entre la vista y el modelo.
Ventajas:

- Facilita el enlace de datos bidireccional.
- Ideal para aplicaciones donde la interfaz de usuario requiere constantes actualizaciones.

Desventajas:

- Requiere más experiencia y puede ser más difícil de implementar correctamente.
- Puede resultar en código más complejo y difícil de mantener en aplicaciones pequeñas.

3.Modelo Vista Adaptador (MVA)

En MVA, el "Adaptador" actúa como una capa intermedia entre el modelo y la vista, transformando los datos del modelo en un formato que la vista pueda entender y presentar adecuadamente.
Modelo: La capa de datos y lógica de negocio.
Vista: Interfaz de usuario que muestra los datos.
Adaptador: Traduce los datos y se asegura de que la vista reciba información en el formato adecuado.
Ventajas:

- Flexibilidad en la presentación de datos sin alterar el modelo ni la vista.
- Útil cuando se deben adaptar datos de diferentes fuentes o APIs.

Desventajas:

- Complejidad adicional al tener que transformar y adaptar datos.
- No es ideal para aplicaciones pequeñas y puede ser difícil de mantener si el adaptador se vuelve muy complejo.

4.Modelo Vista Presentador (MVP)

En MVP, el "Presentador" es el intermediario que maneja toda la lógica de presentación y controla el flujo de datos entre la vista y el modelo.
Modelo: Representa los datos y la lógica de negocio.
Vista: La interfaz de usuario, que en MVP es más pasiva y solo muestra la información que le pasa el presentador.
Presentador: Controla los eventos de usuario, maneja la lógica de presentación, y actualiza la vista en consecuencia.
Ventajas:

- Mayor control sobre la lógica de presentación.
- Facilita las pruebas de unidad, ya que el presentador puede probarse independientemente de la vista.
- Separa de manera eficiente la lógica de negocio de la interfaz.

Desventajas:

- Puede volverse difícil de manejar en aplicaciones grandes debido al aumento de la complejidad del presentador.
- Requiere una estructura de código más organizada para evitar que el presentador se sobrecargue.

5. Arquitectura en Capas

Divide la aplicación en varias capas independientes, donde cada capa tiene una responsabilidad específica:

- Presentación: Interfaz de usuario; maneja la interacción con el usuario.
- Lógica de negocio: Procesa las reglas y operaciones específicas de la aplicación.
- Persistencia: Gestión de la base de datos o cualquier sistema de almacenamiento.

Ventajas:

- Facilita el mantenimiento y escalabilidad.
- Cada capa puede evolucionar de forma independiente.
- Promueve una clara separación de responsabilidades.

Desventajas:

- Mayor complejidad de gestión y comunicación entre capas.
- Puede ser un reto mantener la eficiencia de la aplicación en proyectos de gran escala.

6. Arquitectura Basada en Eventos

En una arquitectura basada en eventos, los componentes de la aplicación se comunican mediante eventos. Cada componente es independiente y responde a eventos específicos, lo que permite que diferentes partes de la aplicación reaccionen a acciones del usuario o cambios en el sistema.
Emisor: Envía eventos al ocurrir ciertos cambios o acciones.
Receptor: Escucha y responde a los eventos de interés.
Bus de eventos: Un canal central que distribuye los eventos entre los emisores y receptores (opcional, pero común en sistemas grandes).
Ventajas:

- Alta flexibilidad y escalabilidad.
- Permite que los componentes trabajen de forma asíncrona.

Desventajas:

- Dificultad en el seguimiento y depuración de eventos.
- Complejidad en la gestión de la sincronización y consistencia de datos.

Link del video